Baltzer Journals

A Contracting Model for Flexible Distributed Scheduling

SANDIP SEN[1], AND EDMUND H. DURFEE[2]

[1] *Dept of Mathematical & Computer Sciences*
University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189
Tel: (918) 631-2985
E-mail: sandip@kolkata.mcs.utulsa.edu

[2] *Dept of Electrical Engineering & Computer Science*
University of Michigan 1101 Beal Avenue
Ann Arbor, MI 48109
Tel: (313) 936-1563
E-mail: durfee@engin.umich.edu

We are interested in building systems of autonomous agents that can automate routine information processing activities in human organizations. Computational infrastructures for cooperative work should contain embedded agents for handling many routine tasks [9], but as the number of agents increases and the agents become geographically and/or conceptually dispersed, supervision of the agents will become increasingly problematic. We argue that agents should be provided with deep domain knowledge that allows them to make quantitatively justifiable decisions, rather than shallow models of users to mimic. In this paper, we use the application domain of distributed meeting scheduling to investigate how agents embodying deeper domain knowledge can choose among alternative strategies for searching their calendars in order to create flexible schedules within reasonable cost.

Keywords: heuristics, meeting scheduling, contract net, search bias, cancellation

Introduction

We are interested in building intelligent, autonomous software agents that can relieve human users from the burden of routine, tedious information processing activities in organizations. As part of that research agenda, we present in this paper results of our investigations in building autonomous agents for the problem domain of distributed scheduling. In particular, we have analyzed in detail the real-life domain of distributed meeting scheduling, in which autonomous, partially cooperative meeting scheduling agents negotiate with each other to schedule dynamically arriving meeting requests [17].

Our approach has been based on developing and evaluating heuristics by which distributed agents can negotiate scheduling of meetings that arrive over time. Several aspects of our problem domain makes it necessary to investigate new heuristics rather than utilizing previous methods developed for traditional project scheduling problems. Most traditional scheduling methods are batch methods, where requirements for a set of tasks are presented to the system, and the goal of the schedule generator is to minimize some metric like tardiness, lateness, completion time, etc. A number of heuristics have been developed which can be used by a single scheduler to schedule a batch of tasks [7, 10, 8]. In our problem, meeting requests arrive dynamically over time and have to be processed on demand. This precludes the use of a batch mode of scheduling and necessitates a reactive mode of scheduling. Some recent results in matchup scheduling handles schedule disruptions by rescheduling on disruption to match up with a preplanned schedule [2]. We are particularly interested in very dynamic scheduling problems where generating a preschedule is infeasible, and reactive scheduling is the only feasible and effective approach to be used.

On another note, we require that the multiple resources required by the tasks are managed by logically and physically distributed agents (no centralized control). In order to schedule a meeting, several distributed decision makers must agree on a time. In trying to schedule a meeting each scheduling agent would like to reveal as little information as possible about the local calendar, the content of which is valued as private information. In the typical project scheduling problem, this would mean that a task requires multiple resources, where the status of each resource is known only to the agent who manages this resource and is not directly known to other agents. Additionally, information about a non-local resource can only be obtained through negotiation with the agent managing that resource. An effective negotiation mechanism will allow agents to converge on a mutually preferred schedule while revealing only minimal calendar information. Traditional scheduling literature does not address the privacy issue, and the status of all resources is assumed to be readily available to the centralized scheduler.

Our domain also entails that a number of meetings are being scheduled concurrently, and new meetings are arriving while previously received meeting requests are being scheduled. This means that the status of local and particularly non-local resources may change dynamically while negotiation over scheduling a meeting request is going on. Traditional optimization or scheduling approaches do not provide any effective mechanisms to address these problems.

To be effective and useful in such domains, autonomous agents need to use a structured information exchange mechanism that helps them quickly identify solutions to individual and global goals, as well as provide the capability to explain their actions to associated users on demand. Our agents achieve these capabilities by using the contract-net protocol [22], which has been used widely to coordinate the activities of agents in distributed problem-solving systems. To effectively implement this protocol in the distributed scheduling domain, we have addressed each of the critical problem-solving aspects of a contracting agent: how to structure and search for contract proposals in the solution space, how much information to exchange to quickly converge the negotiation process without incurring excessive communication cost, how to bid effectively against announced contracts, how to represent and reason about tentative proposals, and when to withdraw past commitments if faced with new contingencies. We have investigated heuristic strategy dimensions to control each of the above aspects of local problem solving [16].

The choice of strategy for considering time intervals for meetings will have numerous effects, including effects on the density of meetings in different parts of the calendar, the likelihood of scheduling future meetings of different types, the costs of scheduling, and the time needed to schedule meetings. More importantly, given targets for calendar densities and limited costs and time for scheduling, a calendar management agent should adapt its strategy choice based on the larger context of what it expects to schedule in the future and what it knows of the calendars of the other agents. These adaptations might not be under the constant supervision of the user, and thus should be made by embedding domain knowledge (a rigorous model of the task) into the agent, rather than trying to capture a superficial model of the user acting in a sample of cases [12].

In this paper, we use a finite state automata model of contracting agents that enables us to precisely represent the processing stages, the resource requirements, and the interaction possibilities between meeting scheduling agents. From the model, we identify different forms of conflicts between scheduling processes that adversely affect the scheduling efficiency of the system. We then present search biases as heuristics to avoid such conflicts, and cancellation and rescheduling mechanisms that resolve conflicts when conflict avoidance is not possible. Our choice of the finite state automata model is motivated by the success of using this model in analyzing communication protocols [1, 4] and in aiding distributed decision-making [5].

1 Problem Specification

Meeting scheduling is a common problem, and we are all familiar with a large number of constraints and preferences that can be used in scheduling meetings.

Our goal in this work is not to solve all possible variants of the meeting scheduling problem. Rather, we would like to concentrate on what we believe is the core problem. Our work has concentrated on addressing this core problem as defined below, and elsewhere [16] we have suggested how our approach can be extended to address additional constraints that one may want to add on to this core problem.

A meeting schedule consists of a group of meetings for a group of persons. Given a set of n meetings and k attendees (hosts and invitees), a scheduling problem is represented as $\mathcal{S} = (\mathcal{A}, \mathcal{M})$, where $\mathcal{A} = \{1, 2, \ldots, k\}$ is the set of attendees and $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ is the set of meetings to be scheduled. A time *slot* is represented as a date, hour pair $\langle D, H \rangle$. A set of contiguous time *slots* is called a time *interval*. A meeting is represented by a tuple:

$$m_i = (A_i, h_i, l_i, w_i, S_i, a_i, d_i, f_i, \mathcal{T}_i),$$

where

$A_i \subseteq \mathcal{A}$, is a set of attendees of the meeting;

$h_i \in A_i$, is an attendee who will host the meeting;[1]

l_i is the required length of the meeting in hours;

w_i is the weight or priority assigned to the meeting;

S_i gives a set of possible starting times on the calendar for the meeting. If $|S_i| = 1$ the meeting is said to be *constrained* (the exact interval to be used for the meeting, if possible, is pre-specified); if S_i includes all physically possible time slots on the host calendar, assuming that it was empty, that can accommodate a meeting of length l_i starting at that slot, then the meeting is said to be *unconstrained*; otherwise, the meeting is *semi-constrained*;

a_i is the meeting arrival time (when host is aware that it has to schedule the meeting);

d_i is the deadline by which the scheduling of the meeting has to be completed;

f_i is the time at which the host finishes processing this meeting request;

$\mathcal{T}_i$ is the time interval for which the meeting m_i is finally scheduled and is represented by an ordered set $\{\langle D_i, H_i \rangle, \langle D_i, H_i + 1 \rangle, \ldots, \langle D_i, H_i + l_i - 1 \rangle\}$, (here D_i gives the date and H_i gives the starting hour for which meeting m_i is scheduled) if the meeting could be scheduled, and by $\emptyset$ otherwise.

Each scheduling process is associated with a personal calendar on which it schedules meetings. The calendar for attendee $j \in \mathcal{A}$, is represented as

$$\begin{aligned} \mathcal{C}_j = & \{\langle D_s, 0, \chi_{s,0} \rangle, \langle D_s, 1, \chi_{s,1} \rangle, \ldots, \langle D_s, \mathcal{L} - 1, \chi_{s,\mathcal{L}-1} \rangle, \\ & \langle D_{s+1}, 0, \chi_{s+1,0} \rangle, \ldots, \langle D_{s+1}, \mathcal{L} - 1, \chi_{s+1,\mathcal{L}-1} \rangle, \ldots, \\ & \langle D_e, 0, \chi_{e,0} \rangle, \ldots, \langle D_e, \mathcal{L} - 1, \chi_{e,\mathcal{L}-1} \rangle\} \end{aligned}$$

where

D_s is the starting date of the calendar;

[1] Note that h_i may be the only member of the set A_i, which allows agents to reserve time intervals for themselves on the calendar.

D_e is the ending date of the calendar;
$\mathcal{L}$ is the number of hours of work per day; and

$$\chi_{x,y} = \begin{cases} i & \text{if } j \in A_i \text{ and } \langle D_x, y \rangle \in \mathcal{T}_i \\ nil & \text{otherwise.} \end{cases}$$

This means a calendar slot for an agent contains a *nil* value unless a meeting is scheduled in that hour for which this agent is an invitee.

A meeting m_i is said to be scheduled when the same time interval is marked with the meeting identifier on the calendars of all attendees of that meeting. More formally,

$$\forall j, j \in A_i, \text{ and } \exists D_y, \forall H_z, \langle D_y, H_z \rangle \in \mathcal{T}_i, \langle D_y, H_z, i \rangle \in \mathcal{C}_j.$$

The specification of the problem also involves the following constraints:

1. Assuming that meeting scheduling takes time, the time the host becomes aware of the meeting must precede the deadline for scheduling it: $a_i < d_i$. Also, if meeting m_i has been scheduled, it must have been scheduled by the deadline ($f_i \leq d_i$) and it must have been scheduled for some time after the scheduling decision has been made ($f_i < \langle D_i, H_i \rangle$ [2]).

2. The time interval for which a meeting can be scheduled has to be contiguous and meetings cannot be split across days. Therefore, $l_i \in \{1, \ldots, \mathcal{L}\}$.

3. For any individual $j \in \mathcal{A}$, let $\mathcal{M}_j \subseteq \mathcal{M}$ represent the set of scheduled meetings that j attends. Then, $\forall x, y,\ m_x \in \mathcal{M}_j, m_y \in \mathcal{M}_j$, and $x \neq y$, $\mathcal{T}_x \cap \mathcal{T}_y = \emptyset$, meaning no individual can attend more than one meeting concurrently.

4. h_i is responsible for scheduling meeting m_i and communicates with every $j \in A_i$ ($j \neq h_i$) to negotiate for a mutually agreeable time for the meeting and is also responsible for finally assigning the meeting time $\mathcal{T}_i$.

5. The basic unit of information that can be communicated between two individuals is a proposal, consisting of at most one date, hour pair. Let Π_{ixy} denote the set of proposals that individual x sent to individual y in trying to arrive at a mutually agreeable time slot for meeting m_i. If $|\Pi_{ixy}| > 0$, then either $x = h_i$ or $y = h_i$.

6. Individual x cannot directly access the personal calendar of individual y, and relies solely on communication to obtain information about it.

[2] $\langle D_1, H_1 \rangle < \langle D_2, H_2 \rangle$ if either $D_1 < D_2$ or $D_1 = D_2$ & $H_1 < H_2$. Other inequalities are defined accordingly.

Two useful extensions to the meeting scheduling problem as defined above can be envisioned. One is to assign rooms for meetings by regarding the room as a pseudo-person and including it in the set A_i for meeting m_i. The other treats A_i as being composed of groups of individuals such that exactly one person in each group is required to attend the meeting m_i. The tuple representing meeting m_i will also include α_i representing the actual attendees of the meeting [23].

Now we would like to emphasize the applicability of our work beyond the narrow domain of distributed meeting scheduling. Note that the above problem definition can be easily changed to reflect a more general distributed task scheduling problem as below: A task schedule consists of a group of tasks to be scheduled using a set of resources managed by a set of agents. Given a set of n tasks and k resources, a task scheduling problem is represented as $\mathcal{S} = (\mathcal{A}, \mathcal{T})$, where $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ is the set of resource managers (one manager per resource) and $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is the set of tasks to be scheduled. A given task maps to a meeting, and a resource manager maps to a meeting scheduling agent in the above problem description. Given this mapping, the techniques developed in this paper to schedule meetings can be easily adapted to schedule dynamically arriving tasks requiring physically and/or logically distributed resources. The limitation of our current approach is that it does not handle constraints between tasks. As such it can be used for distributed scheduling of unrelated tasks only. We are currently working on enhancing our model to process constraints between the tasks.

2 Scheduling through Contracting

The agents use this representation as they engage in a distributed scheduling process based on the multistage negotiation protocol. The multistage negotiation protocol [6] is an extension to the contract-net protocol that allows for multiple rounds of negotiation before an agreement is reached between the negotiating agents. The protocol involves the following steps. On receipt of a meeting to schedule, the meeting's host searches its calendar for possible time intervals, and proposes the top n ($n \geq 1$) to invitees. Obviously, the larger n is, the more information about its own schedule it is revealing, but the more likely it is that it will successfully schedule the meeting in this round of the protocol (quantitative measures for this and subsequent tradeoffs are given in [18]). The goal is to reveal as little information while successfully scheduling the meeting. The agents have to balance the tradeoff between revealing private information and quickly finding an acceptable time to meet. Agents can be assumed to be only partially cooperative, i.e., invitees will respond only to proposals from the host, and will not be the first to volunteer local calendar information. Apart from serving the privacy concern, this also allows agents to concurrently process multiple meeting requests. If an agent was to transmit its entire calendar to another agent, it would have to wait for the latter to finish processing before it can do any further processing with it.

An invitee, upon receiving a meeting announcement, will return a bid. The bid can either respond *yes/no* to each of the n proposed times (we call this a **yes_no** bidding strategy), or it can respond with m possible meeting times, where those times might overlap with the original n but can also counterpropose new *alternative* time intervals (we call this the **alternatives** bidding strategy). The host, after receiving bids, can attempt to confirm an agreeable time if all of the agents have indicated that a particular time is free for each of them, and otherwise the host will repeat the process with a new announcement message giving a new selection of n meeting times.

2.1 DMS Process Model

In this section, we develop a finite-state automata model of the meeting schedule processes. The purpose of the model is to succinctly represent the multistage negotiation protocol used by the processes to schedule meetings, as well as to identify the sources of interaction between different scheduling processes.

To model the DMS processes for this task, we have extended the finite automata framework developed by Casavant and Kuhl [5] for modeling communicating processes. To apply their framework, we have had to assume that the graph representing individuals as nodes is fully connected (which means that each individual can talk directly with any other individual without going through a third party). In the actual network implementation of our system, the scheduling agents communicate with each other using electronic mail messages. Since the processes are directly communicating with each other and messages are not being relayed, from a logical point of view they are fully connected. If another connection topology is used, however, the only effect on the system will be minor reduction in throughput as long as the messages are delivered to their recipients without any appreciable delays.

Casavant and Kuhl [5] represents each node by a finite automaton. This hides the details of local decision-making and how different meeting requests for a single individual compete for resources. Therefore, the assumptions they make in their model are inadequate for our domain. We circumvent this by representing meeting m_i by $|A_i|$ processes, one for each attendee of the meeting. The process corresponding to the meeting m_i at attendee j is modeled by the finite automaton (FA) M_{ij}, whose formal definition is as follows:

$$M_{ij} \;=\; (Q(i,j), \Sigma(i,j), \Delta(i,j), \delta_{ij}, s_{o_{ij}})$$

where

$$
\begin{aligned}
Q(i,j) &= Q_{int}(i,j) \times Q_{ext}(i,j) \times \Re(i,j) \times P(i,j) \times P_r(i,j)^z;\ z = |A_i| \\
\Sigma(i,j) &= P_{rin}(i,j)^z \times Q_{rext}(i,j)^z;\ z = |A_i| \\
\Delta(i,j) &= P(i,j) \times Q_{ext}(i,j) \\
\delta_{ij} &= Q(i,j) \times \Sigma(i,j) \longrightarrow Q(i,j) \\
s_{oij} &= \text{initial state of } M_{ij}
\end{aligned}
$$

and: Q_{int} represents the internal component of the state of M_{ij} not accessible to any other process; Q_{ext} represents the component of state exchanged with other processes; $\Re(i,j)$ represents the calendar resources that can be used by the process (not exchanged with other processes); $P(i,j) \in \{\wp_0, \wp_1, \ldots, \wp_d, \wp_{d+1}\}$ represents the current *phase* or the status of the scheduling process corresponding to meeting m_i for attendee j; $P_r(i,j)^z$ is the process M_{ij}'s knowledge about the phase of each of the other processes involved in scheduling meeting m_i; $P_{rin}(i,j)^z$ is the actual current phase of each of the other processes, that is, part of the present input to the process M_{ij}; and $Q_{rext}(i,j)$ represents the most recent knowledge M_{ij} has about the states of the other processes scheduling meeting m_i.

A process M_{ij} will interact with other processes in two possible ways. First, because attendees of the same meeting must pass information around to converge on a time, M_{ij} will interact via communication with the other M_{ik}—the processes for other attendees of the same meeting m_i. Second, because the same user has separate processes for scheduling the different meetings he or she will attend, M_{ij} will interact with other M_{lj} for other meetings m_l—this interaction (we prefer to refer to this form of interaction as conflict) takes place in the processes' contention for the shared calendar.

2.2 Interaction Via Communication

One type of critical interaction arises through communication between processes for different attendees that are trying to schedule the same meeting. As mentioned before, the constraints on the problem allows explicit communication between individuals as the only means by which information can be acquired to develop more complete global knowledge. This type of interaction then plays a crucial role in the quality of the local decision-making procedures which utilize the local view of global knowledge. Modeling this type of interaction is achieved through the use of phases and properly specifying the set of states that the finite automata representing processes can have, together with precisely formulating the state transition functions.

The following represents the complete set of states for a DMS process using our protocol:

$$\mathcal{Q} = \{\mathcal{Q}_d, \mathcal{Q}_g, \mathcal{Q}_a, \mathcal{Q}_r, \mathcal{Q}_v, \mathcal{Q}_f\}$$

where

- $\mathcal{Q}_d$ is the *decision* state, which assesses the proposals generated and received

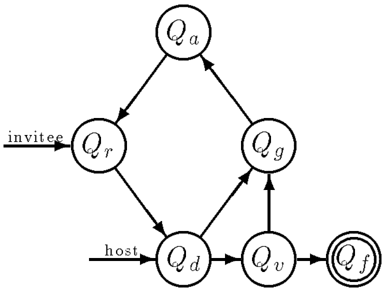

Figure 1: A finite state automaton representing the meeting scheduling processes. A host process starts in state Q_d while an invitee process starts in state Q_r.

so far, and decides either that a mutually proposed interval exists and should be verified, or that new proposals should be generated.

- Q_g is the *generative* state, in which a process produces either proposals or bids (depending on whether it represents the host or an invitee of the meeting).
- Q_a is the *announce* state, in which a process actually communicates proposals or bids to other cooperating processes (processes involved in scheduling the same meeting).
- Q_r is the *receive* state, in which a process waits for proposals or bids to come in from other cooperating processes.
- Q_v is the *verification* state, where a process evaluates whether a mutually proposed time interval can be actually used for scheduling the meeting.
- Q_f is the *final* state, which a process enters when it succeeds or fails to schedule a meeting.

The finite state automaton corresponding to host and invitee processes for a meeting is presented in figure 1. Note that the host process for a meeting starts out in state Q_d, while invitee processes for the same meeting start out in state Q_r.

A process in any state has a phase that represents the current information exchange with other processes. After a number of phase changes, the process might change its externally-known state Q_{ext}, which can prompt a change in the states of other processes interacting with this process. Let us examine the components of process phases in a little more detail to capture the semantics of the process. The degree of information exchange at a particular state is represented as the subscripts of the phase components. The set of transitions $\Gamma = \{\wp_0 \rightarrow$

$\wp_1, \ldots, \wp_{d-1} \rightarrow \wp_{d}\}$ represents the accumulation of more and more information by a process. Actually each such transition is triggered either by the arrival of a proposal message from another process or by some local decision-making. The transition $\wp_d \rightarrow \wp_{d+1}$ reflects the decision which will either result in re-initiation of information gathering at the same state (transition $\wp_{d+1} \rightarrow \wp_0$) or a change of state of the process. Whereas phase transitions are used by Casavant and Kuhl [5] only to represent exchange of information between modules, we use phase transitions to additionally represent progress in local information processing.

We will use two examples to clarify the usefulness and necessity of phases in representing the scheduling processes:

1. Consider a host process that has just sent messages to the other attendees proposing possible meeting times. The host process is in state $\mathcal{Q}_r$, waiting to receive information back from the invitees. As messages come to it, the host changes phases (such as from the phase of waiting to hear from i attendees to waiting for $i-1$ attendees). These phase transitions constitute the set Γ as mentioned above. The final phase transition for a state (such as hearing from the last attendee) causes the process to change states (such as moving into state in which it evaluates the messages it has received). This transition corresponds to the transition $\wp_d \rightarrow \wp_{d+1}$ as mentioned above.

2. Consider a host (invitee) in the *generative* state $\mathcal{Q}_g$. It is trying to produce n proposals (bids). After generating the ith of the first $n-1$ (for $n > 1$) such proposals (bids) there is a phase change from $\wp_i \rightarrow \wp_{i+1}$ within state $\mathcal{Q}_g$. When the last proposal (bid) is generated, the phase transition $\wp_n \rightarrow \wp_{n+1}$ results in a change of state of the process to the *announce* state $\mathcal{Q}_a$.

2.3 Conflict through Shared Resources

In this section, we identify different modes of interaction between two processes that are sharing the same resource (an agent's calendar) to schedule different meetings. Scheduling inefficiency arises due to interaction via shared resources, when two such processes try to use overlapping time intervals to schedule their respective meetings.

The particular scheduling strategy choices that play a major role in affecting these types of interaction are the choice of the *commitment strategy* and the choice of *search biases* in scheduling meetings. While commitment strategies affect the frequency of conflicts between concurrently active scheduling processes, search biases affect the frequency of conflicts between new processes and processes which have already finished scheduling their meeting requests.

The choice of committing or not committing to a proposed time interval amounts to either blocking or not blocking valuable calendar resources until complete agreement is reached. Commitment can cause non-optimal schedules as some meetings block time intervals that cause other meetings to be abandoned due to lack of uncommitted times within the meeting's constraints. In some instances, those blocked intervals might later be released. On the other hand, blocked time intervals prevent attempts to propose overlapping time intervals for two different meetings, which can save scheduling time and the amount of information exchanged to schedule meetings. So, although the primary effect of commitment is on the success of scheduling meeting requests, this strategy choice also affects the total

We can formally represent the resource requirements of process M_{ij} as a 4-tuple,

$$\Re(i,j) \;=\; (v_{ij}, p_{ij}, b_{ij}, r_{ij})$$

where

- v_{ij} represents the set of all *viable* time intervals that could have been proposed for meeting m_i by individual j. It is given by the set of all time intervals of length l_i whose starting slot belongs to S_i (S_i is the set of possible starting times on the calendar for meeting i).

- p_{ij} represents the set of time intervals that have been *proposed* by individual j and are still being considered for meeting m_i. $p_{ij} \subseteq v_{ij}$, since only viable time intervals are proposed.

- b_{ij} represents the set of time intervals that have been *blocked* for probable use by individual j for meeting m_i. These time intervals are under active consideration, but at most one of these will be used for the meeting. $b_{ij} \subseteq p_{ij}$, since only a subset (possibly empty) of the proposed time intervals can be blocked.

- $r_{ij} \;=\; \mathcal{T}_i$ if m_i has been scheduled (represents the time *reserved* for the meeting), and is $\emptyset$ otherwise. $r_{ij} \subseteq p_{ij}$, since the finally reserved time interval for a meeting is one on which all attendees have agreed and hence must have been proposed.

For process M_{ij}, v_{ij} represents the static part of resource requirement, p_{ij} and b_{ij} are the dynamic parts, and, assuming no cancellation, r_{ij} changes at most once (from $\mathcal{T}_i = \emptyset$ to $\mathcal{T}_i \neq \emptyset$ if $\rho_i = 1$) during the lifetime of the process.

The following kinds of conflicts can take place between two processes $M_{x,j}$, $M_{y,j}$, $x \neq y$ in which an individual j is participating:

possible: $\exists X, Y,\ X \in p_{xj}, (\exists y, Y \in p_{yj}), X \bigcap Y \neq \emptyset$. If overlapping time intervals have been proposed for different meetings by the respective processes, there is a possibility that both these meetings could be scheduled for these time intervals in which case one of the processes will fail to schedule its meeting.

actual: $\forall X, \exists Y, X \in v_{xj}(\exists y, Y \in r_{yj}), X \bigcap Y \neq \emptyset$. This scenario corresponds to the case where a request for a meeting m_x comes in such that all viable time intervals corresponding to that meeting overlap with reserved time intervals for some other meetings (m_y). In such a case, the processes M_{xj} and M_{yj} are actually competing for overlapping intervals of time and this results in a failure to schedule meeting m_x.

preemptive: $\exists X, Y, Z, X \in v_{xj}, (\exists y, Y \in b_{yj}, X \bigcap Y \neq \emptyset) \bigwedge \neg(\exists z, Z \in r_{zj}, X \bigcap Z \neq \emptyset)$. This scenario corresponds to the case where a request for a meeting m_x comes in such that at least one viable time interval corresponding to that meeting overlaps with a blocked time interval for some other meeting (m_y), but does not overlap with any reserved time intervals. If $\forall X, \exists Y, Z, X \in v_{xj}(\exists y, Y \in b_{yj}, X \bigcap Y \neq \emptyset) \bigwedge \neg(\exists z, Z \in r_{zj}, X \bigcap Z \neq \emptyset)$, it is not possible to schedule m_x, which affects the success ratio of the scheduling strategy.[3]

From the above analysis we can see that scheduling efficiency can be increased by using heuristic strategies that eliminate or restrict harmful interactions between concurrently active scheduling processes working on different meetings. In a previous paper, we have argued on using adaptive commitment strategies to reduce preemptive and possible interactions between scheduling processes [21]. In the following section, we experimentally evaluate alternative search biases that reduce the frequency of the different kinds of conflicts mentioned above. But no search bias can guarantee the elimination of all conflicts. This means that under certain circumstances, we have to resort to canceling previously scheduled meetings to accommodate newly arrived meeting requests. We present such a cancellation algorithm in Section 4.

3 Avoiding conflicts through proper Search Bias

Our earlier work [18] tacitly assumed that meetings should be scheduled as early as possible, and yielded solutions where calendars tended to be dense at the beginning and sparse at the end. This "lumping" has the disadvantage that it places uneven demands on the user, and that it cannot easily accommodate what we call high-priority short-notice (HPSN) meetings. The latter means that, if the user suddenly learns of a meeting that must be scheduled very soon, there tends to be no open slots for this meeting, leading to costly rounds of cancellation and rescheduling.

[3] Note that the scheduling process does not wait to see if the blocked time interval is actually used or not, but simply signals a failure to schedule the new meeting. This design decision was incorporated to prevent deadlocks. Of course, the process could time out after some prespecified time period, but then the scheduling process will slow down considerably. Various other implementation "hacks" can also be used to alleviate this problem.

Hence, this search bias gives rise to high actual conflicts for HPSN meetings.

More evenly loaded calendars are likely both to place more steady demands on users and to accommodate HPSN meetings more easily. Building such a calendar means that, when scheduling a new meeting, preference should be given to slots that are in the least dense portion of the calendar. Unfortunately, search with this criterion is not nearly as systematic as with the previous (prefer early) criterion, because, while all agents will agree on what is earlier or later (so that they all progress through their calendars the same way), the agents will generally disagree on where the least dense parts of the calendar are, since each has a different calendar with different meetings distributed in it.

Moreover, while the resulting schedule will more likely be able to accommodate HPSN meetings, it will have more trouble scheduling long meetings. That is, scheduling meetings in less dense parts of the schedule tends to fragment available time into smaller and smaller pieces: even if an agent has n hours of free time, those hours might be broken into short spans across several days. Hence, this search bias gives rise to higher actual conflicts for long duration meetings. Note that a preference for earlier meetings would not have this problem, since it will tend to leave longer contiguous blocks of time at the end of the calendar.

The upshot is that the decision about what to value most in a solution depends on factors including: preference for evenly-loaded schedules versus wanting to get things out of the way early on; likelihood of needing to schedule HPSN meetings in the future versus likelihood of needing to schedule long meetings in the future; and desired systematicity in the scheduling process itself, with resulting implications for computation and communication overhead. In the rest of the paper, we go beyond these untried qualitative statements of tradeoffs, and focus on validating and quantifying them to develop guidelines for choosing among them for adaptive surrogate agents.

3.1 Types of Search Bias

How should we bias the distributed search process to yield desirable solutions? We view a search bias as an *a priori* measure of the goodness of certain particular solutions being evaluated. In this respect, different search biases can be related to different *Value Goodness* measures [11], as used in the constraint satisfaction literature. We now present three different search biases, identifying how they work, and the types of solutions (calendar profiles) they generate:

Linear early (LE): This search bias was used in our earlier work to produce calendars with a density profile (distribution of meetings) where the number of free intervals of any given length is very low close to the current date on the calendar, and increases steadily as we look further down. With this search bias, an agent attaches increasing goodness values to intervals earlier in the calendar, i.e., given a meeting, the agents try to schedule the meeting as early as possible. The search bias is implemented by making an agent start searching the calendar at the earliest possible scheduling opportunity, skipping over any intervals overlapping with already scheduled meetings, and negotiating with the earliest free interval on the calendar long enough to accommodate the meeting. This search bias can be likened to the *First Fit* memory allocation scheme [15].

Linear least dense (LLD): The resulting schedules produced with this search bias will have a density profile in which the number of free intervals for any given length is approximately constant across the calendar length. With this search bias, an agent tries to schedule a meeting in the least dense part of its calendar. The search bias is implemented by the host of the meeting ranking all the empty intervals on its calendar long enough to accommodate a given meeting (and within the window of acceptable times for the meeting) by a function that measures the number of free calendar slots around that interval. The agent then steps down this ranked list and negotiates with other attendees of the meeting until it can schedule the meeting. This search bias can be likened to a local form of *Worst Fit* memory allocation strategy [15].

Hierarchical (H): This is another method of producing even density profiles across the calendar. With this search bias, an agent tries to schedule a meeting in the least dense part of the combined search space of all the attendees of the calendar. The search bias is implemented by building an abstraction hierarchy atop the linear calendar for each meeting-scheduling agent. At each node in the hierarchy, agents keep a record of the number of intervals of different length free below that node in the hierarchy. The calendar space lends itself to a very natural hierarchy of hours, days, weeks, etc., and the agents participating in a meeting can first identify a good week to meet in, then identify a good day within that week, and finally an actual interval within that day. Given a meeting of some particular length to schedule, the host asks for and receives information from all the invitees about how many intervals of that length are open at each node (e.g., at each week) of the highest level of the hierarchy. It uses these numbers to compute the probability of scheduling the meeting under each of these nodes, ranks the nodes, elaborates the best one, and proceeds to repeat the process for the next level of the hierarchy under the elaborated node. At the ground level, information exchange takes place like the **LE** scheme. Backtracking occurs if a particular portion of the ground level being elaborated contains no solution to the scheduling problem. In this paper, the levels of the hierarchies used by the agents correspond to days and hours only. Agents first negotiate about the likelihood of scheduling a meeting on different days, then choose the most likely day and start **LE** negotiation within that day. The host may fail to schedule the meeting in the most likely day, and will then backtrack to the next most likely day. This search bias can be likened to a global form of *Worst Fit* memory allocation strategy (in the sense that the fitting takes place in the most free part of the combined search space of all the participants of a meeting).

3.2 Expectations

To verify that these biases do in fact perform as we anticipate, we need to be clearer about exactly what we expect. To do this, we have to clarify what we should measure to assess the performance, both in terms of achieving acceptable solutions and in terms of computational and communication costs.

The evaluation metrics that we will be using are the following:

Communication cost (CC): Communication cost is measured by the average number of information packets exchanged to schedule a multi-agent meeting. One information packet consists of a proposal from a host or an invitee. In the case of **H** search bias, while negotiating at a non-leaf node of the hierarchy, a node-number and meeting-likelihood pair is considered an information packet. This measure provides us with an estimate of the amount of bandwidth required to schedule meetings.

$$C = \frac{\sum_{i=1}^{n} \sum_{j=1}^{k} c_{ij}}{n_m}$$

where $n = |\mathcal{M}|$, $k = |\mathcal{A}|$, and

$$c_{ij} = \sum_{\forall y, y \in A_i \text{ and } j \neq y} |\Pi_{ijy}|,$$

where Π_{ixy} denotes the set of proposals that individual x sent to individual y to schedule a meeting m_i, and n_m is the number of multi-agent meetings scheduled. This measure is used to distinguish otherwise equivalent biases by preferring those which requires less communication bandwidth.

Iterations (I): Iterations required to schedule a meeting is measured by the average number of rounds of negotiation entered into by the participating agents in a meeting before a meeting is scheduled, or it is recognized that the meeting cannot be scheduled. This measure indicates the amount of time required to schedule a multi-agent meeting.

Slots Searched (SS): Slots searched to schedule a meeting is measured by calculating the sum of the number of possible intervals on the calendar looked at by the participants while trying to schedule a meeting. The measure is an average over all the meetings scheduled. It is indicative of the search complexity (and correspondingly, the time taken) for finding intervals to propose.

Meeting Hours Missed (MHM): Meeting hours missed represents the success of scheduling the meetings requested so far, and is calculated as the number of requested meeting hours per agent that could not be scheduled.

$$\text{MHM} = \frac{\sum_{i=1}^{n} l_i * |A_i| * (1 - \rho_i)}{|\mathcal{A}|},$$

$$\text{where } \rho_i = \begin{cases} 1 & \text{if } m_i \text{ has been scheduled} \\ 0 & \text{otherwise.} \end{cases}$$

Density Profile Characteristics (DPC): The density profile characteristics are plots that display the variation of the number of free intervals of different length over the length of the calendar. The numbers are averaged over all the agent calendars. This measure is indicative of the spread of the meetings scheduled on the calendar, and can be used to predict the success of scheduling different kinds of meetings on the given calendar.

The first three of these criteria measure the cost incurred in the process of scheduling the meeting, whereas the last two criteria reflect the quality of the schedule generated both in terms of how well it has scheduled known meetings and how likely it is to schedule future meetings. We now briefly present our expectations of how different search biases will perform on the different evaluation criteria, and also provide brief intuitions behind these expectations.

The **LE** search bias should have a negligible **MHM** value so long as all the meeting requests received have a large window for scheduling. This is because the search bias tends to lump the meetings together thus leaving space for other meetings to be scheduled at the end of the calendar. By selecting the *alternatives* **bidding** strategy, large portions of the search space not containing solutions to the scheduling problem can be quickly pruned [17], leading to inexpensive **I** and **CC** measures. As the number of meetings scheduled grows, more and more intervals need to be looked at before finding a free interval, and hence the **SS** measure would become increasingly costly. In practice, we can reduce the local search cost by caching open intervals indexed by meeting lengths. If proposed time intervals are committed or when meetings are canceled, however, the indexes have to be continually updated. This incurs additional overhead, and should be added to the processing cost incurred by the **LE** search bias. From what we have discussed before, it can be inferred that with the **LE** search bias, the density profile characteristic curves will be low at the start of the calendar and rise towards the end of the calendar.

The **LLD** search bias fragments the search space as it places meetings in the least dense parts of the calendar. Therefore, as the number of hours scheduled $\mathcal{H}$ approaches the total length of the calendar L, it is more likely that with the **LLD** search bias some meetings (particularly long ones) cannot be scheduled, resulting in **MHM**> 0. This reasoning also holds for the **H** search bias. The **LLD** search bias should be more expensive in iterations compared to the other two search biases because counterproposals from invitees cannot help to eliminate portions of search space from consideration. The **SS** measure is expected to be extremely high as all possible slots on the host's calendar are looked at and ranked before negotiation begins[4]. The density profile characteristics produced by the **LLD** search bias should be even across the calendar.

The **H** search bias adds communication cost and iterations during the negotiations at the non-leaf nodes of the hierarchy. The added communication cost is considerable because for each node in the hierarchy, the host has to receive information for each of the nodes below that node from all the invitees (e.g., for a given week, the host needs information about each day in that week from all of the invitees; since information about only one node is sent in a packet, this implies that a lot of packets have to be communicated). The expected benefit is quickly identifying portions of the search space that are highly likely to contain a solution. It would be instructive to see whether the one-time cost of information exchange at abstract levels can be offset by efficient searching of the ground space. **MHM** values should be similar to the **LLD** case. The slots searched using this search bias should be few, as the information contained in the hierarchy quickly helps focus on mutually free intervals on the calendar. The density profile characteristics should be smooth across the calendar. If individuals were not reserving times for themselves on the calendar, the smoothness of calendars produced by the **H** search bias would have been more than the smoothness of calendars produced by the **LLD** search bias. This is because with the **LLD** search bias the meeting is placed in the most free part of the host's calendar. Since in a group each agent is more often the invitee rather than the host to a meeting, an individual agent does not get sufficient opportunity to smooth out the calendar. Using the **H**

[4] These intuitions match closely with the findings of researchers in operating systems, where the *First Fit* memory allocation strategy has been found to produce faster search (corresponding to less **SS**) and better storage utilization (maps to less **MHM**) over the *Worst Fit* memory allocation strategy [15]. Here also we can use caching methods to reduce the cost of local search, but the cost of bookkeeping can become noticeable as in the case of the **LE** search bias described above. Note though, in this work, we are more concerned about a number of other criteria including the ability to generate desired density profile characteristics, and hence the **LE** bias is not always preferable to the **LLD** or **H** search bias.

search bias, however, every agent to a meeting has the same influence on where the meeting gets scheduled. Averaged over a number of meetings, this allows agents to generate evenly loaded calendars. In our experiments, however, agents can schedule meetings with themselves, and this allows even the **LLD** search bias to smooth out calendars as well as the **H** search bias.

3.3 Experimental Results and Analysis

In this section we first describe the parameters used in setting up the experiments, then present the experimental results obtained by running experiments with the different search biases on identical meeting requests.

We report results from two different sets of experiments involving two different groups, where a group is characterized by the number of individuals scheduling meetings and the length of their corresponding calendars. For the smaller group, we consider three agents with 5 day calendars. The corresponding numbers for the larger group are 10 and 14 respectively. In both cases, each day consists of 9 hours. The agents start with an empty calendar and are given a number of meetings to schedule. The meeting requests are assigned such that if all the meeting requests get scheduled, each of the agents will have $\mathcal{H}$ hours reserved in their calendar. Given a total number of L =45 (126) hours on each agent's calendar for the smaller (larger) group, we have run experiments varying $\mathcal{H}$ from 10 up to 40 (70 up to 130), in steps of 5 (10). All meetings are totally unconstrained in that it is acceptable to schedule any of them in a free interval anywhere over the length of the calendar[5]. For each search bias, and for each value of $\mathcal{H}$, the results reported are averaged over 1000 runs of randomly generated meetings. The meeting lengths are chosen from a discrete probability distribution assigning the probabilities 0.3, 0.25, 0.2, 0.15, 0.05, 0.05, 0.0, 0.0, and 0.0 to meeting lengths 1, 2, 3, 4, 5, 6, 7, 8, and 9 hours respectively (this distribution is somewhat representative of meetings in real life; shorter meeting lengths are much more common than meetings that occupy most of the day). The number of attendees for a meeting is picked using a uniform distribution. In the simulation, we schedule meetings one at a time, that is, a host is given a meeting to schedule, and only after the scheduling process for that meeting is terminated, do we present the next meeting to be scheduled to an agent. Though one of the primary benefits of a distributed formulation of the scheduling problem is the increased throughput obtained by concurrent processing of multiple tasks, for the following experiments we use a sequential mode to identify the effects of search bias independent of concurrent processing issues. The performance measures reported are those mentioned in Section 3.2. In the following, we will briefly highlight our findings from experiments with the

[5]This means that there is no particular bias towards either HPSN meetings or for meetings in the future. This is because the meetings can be scheduled anywhere on the calendar.

smaller group, followed by a more detailed analysis of the experiments with the larger group.

For the smaller group, the number of iterations and the communication cost obtained with the **LE** bias is low and varies little with varying $\mathcal{H}$. Communication cost is roughly 3 times the number of iterations, because multi-agent meetings can involve either 2 or 3 persons, and the corresponding communication cost per iteration of the scheduling process is 2 and 4 respectively (more generally, for a k person meeting, the communication cost per iteration of the normal information exchange phase of our negotiation protocol is $2(k-1)$). Low values of these metrics can be attributed partly to the use of an effective **bidding** strategy (*alternatives* option). The fact that the **LLD** bias performed only slightly worse on these metrics indicates that other reasons contributing to these results include the relatively empty calendars available to the agents to process meeting request, the small number of agents, and high percentage of multi-agent meetings (which means that the calendars fill up similarly). This analysis prompted us to experiment with larger groups to see how these biases scaled up. The performance of the biases on other metrics were similar for the small and the large group; for these results we refer the readers to the following discussion of our experiments with the larger group. Another noteworthy observation from this set of experiments was that **LLD** bias took fewer iterations to schedule meetings than the **H** search bias while producing similar DPCs. So, for small groups, the extra iterations in descending the hierarchy with the **H** search bias is not recuperated from more effective search of the joint calendar space. Because of the relatively small search spaces, the **LLD** search bias seems to be able to produce desired DPCs without losing out on other performance metrics.

Results from experiments with the larger group are presented in Figure 2 and Figure 3, which display the performance of the different search biases on the metrics evaluating the actual scheduling process and the metrics evaluating the resulting calendar density profile respectively. The latter presents the results of experiments with $\mathcal{H} = 100$ (graphs for other values of $\mathcal{H}$ have similar characteristics). In these experiments, 75% of the meetings required the attendance of two or more agents.

In sharp contrast to the results obtained with the smaller group, in experiments conducted with the larger group, the **H** search bias schedules meetings quicker than the **LE** or **LLD** search biases. This is due to the fact that the **H** search bias can better handle disparities between attendee calendars. When attendee calendars are stacked up differently, the **H** bias use the hierarchical information exchange to quickly identify parts of the calendar that are likely to contain a mutually acceptable interval. The other search biases waste more time eliminating unlikely candidates from the search process. This is particularly true for the **LLD** search bias, in which case the host does not receive informative replies from the invitees to aid the negotiation process. The number of iterations taken to schedule meetings with the **H** search bias is almost constant over $\mathcal{H}$. One of these iterations is spent to gather information about the density profiles at the day level of the hierarchy from the invitees of a multi-agent meeting. This particular iteration also incurs a heavy communication cost as the number of information packets sent by each invitee is equal to the number of days in the calendar, as opposed to sending only one packet for any other iteration. As such, the communication cost is considerably higher than in the other two cases.

When we look at the slots searched metric for the **LE** bias, we see a linear increase with $\mathcal{H}$; this was anticipated because with more hours full on the calendar, the agent has to search progressively further to find an empty interval on the calendar. As expected, the slots searched by the **LLD** bias is extremely large: all possible intervals are looked at to determine if they can accommodate the given meeting. Only the host is involved in this ranking of plausible intervals; otherwise the value for this metric would have been much higher. The number of slots searched, on the other hand, is extremely small as the information maintained in the hierarchy helps in quickly identifying potential intervals to be used for a meeting. The values for this metric grows slightly with increasing $\mathcal{H}$ to reflect increasing search at the ground level by an invitee to come up with an *alternative* bid (or counter-proposal) to a proposal from the host, as more and more hours on the calendar are reserved for other meetings.

The observed metric of meeting hours missed with different search biases is as expected, since agents start failing to schedule meetings only when $\mathcal{H}$ is close to L. Most of the meetings missed are either multiple agent meetings or are long meeting requests arriving at the end of the scheduling run. There is no significant difference in the performance of the **H** and **LLD** search bias on this metric. But using the two-sample t procedure [13] we find the performance of these two search biases and the **LE** search bias is significantly different at least at the 99% confidence level under all load conditions. The value for this metric is noticeably greater for the **LLD** and **H** bias than that for **LE** bias for $\mathcal{H}$ ranging from 80 to 120. In its attempt to evenly schedule meetings across the calendar length, the **LLD** search bias ends up fragmenting the calendar space, and hence is unable to accommodate a percentage of long meetings as $\mathcal{H}$ increases above a threshold value. An interesting result of our experiments with the two groups was that, though the **LE** search bias leads to smaller values of meeting hours missed in the larger group, the proportional savings obtained are far less than in the smaller group. This is because even though each agent is relatively free towards the latter part of their schedules when using the **LE** bias, individual schedules can differ significantly in the exact intervals that are free in these portions. As such, even with space compaction, short meetings with large numbers of attendees may not get scheduled.

Figure 3 shows that when using the **LE** bias, more and more intervals of any given length are open on the calendar as one proceeds from the front to the end of the calendar. This means that, as per expectation, this particular search bias is scheduling most of the meetings grouped together near the start of the calendar. On the other hand, the **LLD** and **H** search biases are able to deliver on the promise of even density profiles across the length of the calendar. Results are better than expected for the **LLD** bias because this bias only balances the host's calendar, but that seems sufficient in balancing each agents' calendar since everyone gets to be the host with equal frequency, and more importantly, they have a number of meetings with themselves which smooth out their respective density profiles.

3.4 Observations

There are several important observations to be made given the results and analysis of the last section; some of these reinforce our expectations, while a few provide new insights to the properties of the search biases. In the following, we highlight our observations for each performance metric. Both **LLD** and **H** search biases generate even density profile characteristics (**DPC**) across calendar lengths in contrast to the skewed density profiles produced by the **LE** search bias. This is an experimental confirmation of the characteristics these search biases were designed to generate. Observations generally true for all the search biases considered are:

1. for any given $\mathcal{H}$, the number of open intervals increases with decreasing length of the intervals (in a free interval that can accommodate a long meeting, we can schedule a number of smaller meetings),

2. the number of intervals open for a given length is a non-increasing function of $\mathcal{H}$ (this is observed by comparing different graphs like Figure 3 with varying $\mathcal{H}$).

On the other hand, the **LE** search bias leaves room for scheduling long meetings, and hence, particularly in cases where $\mathcal{H}$ is close to L, results in fewer meeting hours missed (smaller **MHM** values) than the **LLD** and **H** search biases. From Figure 3 we find that the number of open slots for longer meetings at the end of the calendar produced by the **LE** search bias is not as high as expected compared to those in the calendars produced by **LLD** and **H** search biases. This is because the **LE** search bias ends up scheduling much more hours on the calendar compared to the other search biases, and hence has less space to accommodate more meetings. The purpose of building evenly dense profiles is to retain the flexibility of scheduling HPSN meetings. If that flexibility is obtained at the price of being unable to schedule some regular, run-of-the-mill meetings, then that is often too high a price to pay. So, depending on the importance of HPSN meetings, one may either want to use the **LLD** or **H** search biases in favor of the **LE** search bias across the range of $\mathcal{H}$ values, or to $\mathcal{H}$ values below a threshold only. The behaviors of the different search biases on the **MHM** match our expectations developed in Section 3.2.

Though meetings can be quickly scheduled (smaller **I**) in small groups by using the **LE** search bias, as we consider larger groups, it is clear that the **H** bias is the most expedient bias for scheduling meetings. Our further studies [20] have shown that, using probabilistic methods, we can build a scheduler that can predict the number of iterations required for each of the search biases given current information about the calendar densities of the participants, allowing dynamic and quantitatively informed adaptation of the search bias over time. Communication cost (**CC**) required to schedule meetings is greater for **H** search bias than the other two (of which, the values for **LLD** are higher than those for **LE** search bias), because of the increased overhead of the first iteration (as information about more days needs to be exchanged). This problem may be solved by multiple levels of the hierarchy; if 10 days were separated into two weeks, and a solution was found in the least dense of the two weeks, communication cost for the 5 days in the other week could be saved.

In terms of the slots searched metric (**SS**), the **H** search bias is a sure winner. This metric measures the amount of time spent in local search by the agents to find suitable time intervals for meetings. In our past work, we have noted that, in situations where concurrent scheduling of several meetings is taking place, greater time spent in local search will lead to increased probability of mutually harmful interactions between scheduling processes that share the same calendar [21]. So, although the significance of this metric is not apparent in this simulation (with no concurrent scheduling), in real-life this metric will greatly impact the other cost metrics.

For all the search biases considered, we notice a decrease in scheduling efficiency (as measured by the meeting hours missed metric) with an increase in the size of the group. However, in larger groups (with more agents, longer calendars), the **H** search bias takes the least number of iterations, provides more flexibility, and does not significantly sacrifice success rate of ordinary (non-HPSN) meetings (as compared to the **LE** search bias). Hence, we prescribe a switch in search bias from **LE** to **H** as the group grows. The use of **LLD** should be limited to situations where, even though the group is small, HPSN meetings are fairly common.

Given the above tradeoffs between the different search biases, it would be useful to have a predictive mechanism to calculate the expected performance of a search bias given the current system state and the nature of meetings to schedule. We have developed a probabilistic analysis scheme that allows an automated scheduler to choose the most effective search bias when asked to schedule a meeting [16, 20].

4 Resolving conflicts through Cancellation and Rescheduling

We now examine some of the critical issues involved in cancellation and subsequent rescheduling of meetings. The contents of this section are our first attempt to address this significant problem. We have not yet been able to develop a satisfactory user-independent means of evaluation of the proposed techniques. Some of the problems in developing a user-independent performance criteria include issues of comparing pareto-optimal solutions, measures of social utility, local versus global preferences, etc. which require further investigation beyond the scope of this paper. Though we are not in a position to present experimental evaluation to validate our contributions, we believe that our initial investigations provide a comprehensive approach to cancellation and rescheduling that utilizes user preferences as well as scheduling costs and expectations.

A meeting may need to be canceled in order to accommodate another meeting (possibly a higher priority one). This will be done only if the new meeting cannot be scheduled at any other time without canceling an already scheduled meeting, i.e., when an actual conflict takes place. A meeting canceled (or more appropriately, displaced) for another meeting will have to be rescheduled for another time interval.

We first briefly outline the actual process of canceling a meeting, and then present the procedure to decide when and what to cancel. When any attendant of a meeting decides to withdraw from its previous commitment to a time interval to a meeting, it sends a cancellation message to the host of that meeting. The host, in turn, broadcasts the cancellation message to all the invitees of this meeting. Simultaneously, the host also starts re-negotiation for scheduling this meeting using the multistage negotiation protocol. We now describe an algorithm that extends the scheduling protocol presented earlier to incorporate cancellation.

The aim of scheduling is to find an empty interval in the calendar for a newly requested meeting, but when no such interval can be found, the new meeting could 'bump' one or more lower priority meetings (which have to be rescheduled). In the first of these phase, that is, while looking for empty intervals on the calendar, the scheduler need not worry about the priority of the meeting, as it tries to schedule all meetings (assuming all requested meetings have a positive priority). If no interval can be found of the required length that is free on each of the attendees' calendar, the host initiates a special *cancellation phase* which involves extensive use of meeting priorities and other relevant knowledge. In this phase, all the viable time intervals for the particular meeting are ranked on the basis of the net utility of such an assignment. The net utility to agent x of scheduling the new meeting in interval I ($\mathcal{U}_{new,I,x}$) is defined as:

$$\mathcal{U}_{new,I,x} \;=\; w_{new,x} - \sum_{j_{I,x}} \left(\mathcal{C}_{j_{I,x},x} \;+\; q_{j_{I,x}} \;*\; w_{j_{I,x},x} \right),$$

where the summation extends over all the meetings that have to be canceled in order to make the interval I free on agent x's calendar, $\mathcal{C}_{k,x}$ corresponds to the expected cost of agent x for rescheduling the kth meeting, $w_{k,x}$ is the priority of the kth meeting to agent x[6], q_k is the probability of failure in rescheduling the kth meeting. The cost of rescheduling a meeting is estimated by taking into account the number of packets exchanged and the total time taken when the meeting was originally scheduled. Every agent also tries to update its knowledge about the density of the schedules of the other agents (this information can be explicitly requested from another agent while a meeting with that agent is being scheduled, or it may be obtained from another agent who has obtained it from the said agent), and this information is used to estimate the probability of failure in rescheduling a particular meeting.

We consider the resource requirement ($\Re(i,j) = (v_{ij}, p_{ij}, b_{ij}, r_{ij})$) of the process at agent j involved in scheduling meeting i. Let us represent by P^1_{ij} the set of all time intervals that were proposed during the first pass by the host. These were those elements of v_{ij} which did not overlap with another time interval, either blocked or reserved for another meeting by agent j. We define the set of viable time intervals that were not proposed in the first phase of negotiation as $P^2_{ij} = v_{ij} - P^1_{ij}$. Elements of the whole set v_{ij} will be considered in the *cancellation phase* for scheduling the meeting i. The strategies for using these time intervals for negotiation with other processes involved in scheduling the same meeting will vary depending on the scheduling strategy followed during negotiating with elements of P^1_{ij}, and also on certain other factors.

Let us assume that one or more of the invitees of the meeting i were using the **alternatives** bidding strategy, and let R^1_{ij} be the set of all distinct time intervals that were either proposed by the host in the first phase of negotiation, or counter-proposed by the invitees in response to proposed time intervals in P^1_{ij}. Let $\hat{R}^1_{ij} = R^1_{ij} \cap P^2_{ij}$ be the set of intervals counter-proposed by one or more invitees in the first phase of negotiation, and which were never proposed by the host during that phase. We now divide $\hat{R}^1_{ij}$ into $e_i = |A_i| - 1$ equivalent classes $\hat{R}^1_{ij}(1), \ldots, \hat{R}^1_{ij}(e_i)$, where $\hat{R}^1_{ij}(x)$ contains those elements of $\hat{R}^1_{ij}$ that were counter-proposed by exactly x invitees. For completeness we define $\hat{R}^1_{ij}(0) = P^2_{ij} - \hat{R}^1_{ij}$, to be those intervals that were neither proposed by the host, nor counterproposed by any invitee during the first phase of negotiation. We also assume that each of the sets $\hat{R}^1_{ij}(x)$, $x = 0, \ldots, e_i$, are ordered by their position on the calendar. We also divide the set R^1_{ij} into similar equivalent classes, where the set $R^1_{ij}(x), x = 0, \ldots, e_i$ consists of those intervals that were proposed by exactly x attendees (including the host) during the first phase of negotiation. Notice that $R^1_{ij}(x) \subseteq \hat{R}^1_{ij}(x)$, $x = 0, \ldots, e_i$, because the latter involves time intervals counter-proposed by the invitees while the former

[6] Note that this formulation allows different priority assignments to a meeting by its attendants.

consists of those counter-proposals as well as intervals that were proposed by the host.

In the second phase of negotiation, the host starts with the set $R^1_{ij}(e_{ij})$ and then successively considers the sets $R^1_{ij}(e_{ij}-1), \ldots, R^1_{ij}(0)$, if the meeting cannot be scheduled with the time intervals in the set under consideration. Investigation of a particular set $R^1_{ij}(x), x = 1, \ldots, e_i$, proceeds as follows. The host first considers members of the set $\hat{R}^1_{ij}(x)$ and then the members of the set $R^1_{ij}(x) - \hat{R}^1_{ij}(x)$ because in the former case one of the agents who need to be convinced about agreeing to a time interval is the host itself (as the host of the meeting it is easier to work around your own schedule than to persuade another agent to change its schedule). The underlying assumption of forming and using these equivalence classes is that it is more difficult to convince m attendees to change their mind about using a slot for a meeting than it is to convince n attendees to do the same (where $m > n$).

For a given set, each member of the set is used to calculate the utility (for the host) of scheduling the new meeting in that slot. We can use two heuristics to choose the time interval to be used for the new meeting. In the first case, we use any time interval for which the sum of the utilities of all the attendees (invitees + host) scheduling the new meeting in that interval is greater than zero. So, in this **globally beneficial** strategy, the meeting is scheduled for a time interval $\mathcal{T}$ such that

$$\sum_{a \in A_i} \mathcal{U}_{new,\mathcal{T},a} > 0.$$

In the second case, we use any time interval for which the utility of each attendant scheduling the new meeting in that time interval is non-negative, and at least for one attendant the utility is positive. So, in this **locally beneficial** strategy, the meeting is scheduled for a time interval $\mathcal{T}$ such that

$$(\forall a \in A_i, \mathcal{U}_{new,\mathcal{T},a} \geq 0) \;\&\; (\exists a \in A_i, \mathcal{U}_{new,\mathcal{T},a} > 0).$$

So, if we are using the **globally beneficial** method, we negotiate with each member of $R^1_{ij}(x), x = 1, \ldots, e_i$, before considering any member of $R^1_{ij}(x-1)$. Using the **locally beneficial** method though, we can reduce the search space at the host, and negotiate with only those elements in $R^1_{ij}(x), x = 0, \ldots, e_i$, for which there is a non-negative utility for the host of scheduling the meeting at that interval.

The host uses a special message to convey to the invitees that a *cancellation phase* is being entered for scheduling the new meeting (this enables the invitees to process proposals in a different manner than in the first phase of negotiation). In this second phase, the host may propose one or more time intervals per round of negotiation, and the invitees reply with their respective utilities for scheduling the new meeting at proposed time intervals (a modified **yes_no** bidding strategy which provides more information than a binary signal). It is useful for the invitees to use a sort of *committed* strategy in this phase, in that they do not schedule more meetings in the interval being considered before hearing back from the host. This serves to keep the utility value of scheduling the new meeting for a particular interval constant while the agents negotiate using that interval. Note, that this *commitment* is not necessary for an interval if the invitee knows that the host is using the **locally beneficial** method of cancellation and if its utility for scheduling the meeting at the particular interval is negative.

In the case of all invitees using the **yes_no** bidding strategy $R^1_{ij} = P^1_{ij}$. We divide R^1_{ij} into equivalent classes and we define $R^1_{ij}(0) = v_{ij} - P^1_{ij}$. We next proceed with negotiation using these equivalent classes as described before.

In the above process, the host stops the search for an acceptable time interval if the deadline for scheduling the meeting arrives, and informs the invitees to abort their respective search processes as well. If the new meeting is scheduled for an interval, all the meetings previously scheduled and found overlapping with that interval are canceled. In certain situations, such cancellations and rescheduling efforts may generate long chains of computation. This chain is bounded though, since we will not try to reschedule a meeting for which the scheduling deadline has passed. Our method also provides an automatic damping mechanism for restricting domino effects of meeting cancellations. Following our method, sooner or later the utility of canceling another meeting will be negative and this stops the chain. We are currently working on developing better utility measures, as well as generating more accurate estimates of the cost and the probability of failure of rescheduling a meeting (to be displaced to accommodate a high utility meeting) based on the effort required to schedule it originally, and the amount of time that has elapsed since that meeting was scheduled.

Conclusions

In this paper, we have argued in general terms for developing surrogate agents that make decisions based on carefully-constructed models of the application domain, and we have more specifically highlighted the importance of retaining flexibility for future resource requests when searching for a solution to a current resource scheduling problem. Flexibility can take many forms, including flexibility to eventually schedule resources for long periods, and flexibility for scheduling resources for brief periods on short notice. Given these alternative criteria for evaluating scheduling decisions, we have implemented and evaluated several approaches for biasing the distributed search process in a distributed meeting scheduling application. From our results, it it clear that naive decisions about scheduling can have significant and lasting impacts on the ability of a schedule to adapt dynamically to new resource requests. From a practical standpoint, this means that, in applications like distributed meeting scheduling, a human user that is tailoring his or her process should at least be aware of the impacts of alternative choices of search bias, and should have the option of allowing the scheduling system to adapt its own strategies as circumstances change.

We also demonstrated the promise of using a natural hierarchical representation of calendars in the search process. Our results indicate that hierarchical distributed search will be increasingly important as the scheduling problems are scaled up to many agents and long schedules.

When search biases cannot avoid conflicts, conflict resolution becomes necessary. We have developed a structured cancellation mechanism that can resolve conflicts by maximizing a utility measure. This procedure is computationally effective and captures the intuitive reasoning used by humans to cancel and reschedule meetings. Though this procedure needs further evaluation, we believe that this is a good start to addressing the complex problem of renegging on past commitments because of new contingencies or opportunities.

Our future efforts will involve addressing broader class of scheduling problems using the techniques developed here. We are particularly interested in the dynamic, concurrent scheduling of a multitude of resource-constrained project networks (possibly managed by different departments in a group) with significant resource interdependencies [3]. We also believe that our approach of distributed, incremental scheduling in a dynamic domain can be successfully applied to a wide variety of scheduling problems on which other AI techniques have been used [14]. In particular, we have showed that our proposed system of distributed contract-based negotiation can be effectively used in a manufacturing environment [19].

Acknowledgements

This research has been sponsored, in part, by the National Science Foundation under a Research Initiation Award IRI-9410180, a Coordination Theory and Collaboration Technology grant IRI-9015423, and by a grant from Bellcore.

References

[1] Alfred V. Aho, J.D. Ullman, and M. Yannakakis. Modeling communications protocols by automata. In *Proceedings of the 20th Symposium on the Foundations of Computer Science*, pages 267–273, October 1979.

[2] J.C. Bean, J.R. Birge, J. Mittenthal, and C.E. Noon. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, May-June 1991.

[3] Collin E. Bell. Maintaining project networks in automated artificial intelligence planning. *Management Science*, 35(10):1192–1214, October 1989.

[4] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, April 1983.

[5] Thomas L. Casavant and Jon G. Kuhl. A communicating finite automata approach to modeling distributed computation and its application to distributed decision-making. *IEEE Transactions on Computers*, C-39(5):628–639, May 1990.

[6] Susan E. Conry, Robert A. Meyer, and Victor R. Lesser. Multistage negotiation in distributed planning. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 367–384. Morgan Kaufman, 1988.

[7] Edward W. Davis and James H. Patterson. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science*, 21(8):944–955, April 1975.

[8] John Dumond and Vincent A. Mabert. Evaluating project scheduling and due date assignment procedures: an experimental analysis. *Management Science*, 34(1):101–118, January 1988.

[9] Jolene Galegher, Robert E. Kraut, and Carmen Egido. *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1990.

[10] S.C. Graves. A review of production scheduling. *Operations Research*, 29:646–675, 1981.

[11] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.

[12] Pattie Maes and Robyn Kozierok. Learning interface agents. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 459–464, July 1993.

[13] David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics*. W.H. Freeman and Company, New York, NY, 1989.

[14] S.J. Noronha and V.V.S. Sarma. Knowledge-based approaches for scheduling problems: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):160–171, June 1991.

[15] James L. Peterson and Abraham Silberschatz. *Operating System Concepts*. Addison Wesley, Reading, MA, 1985.

[16] Sandip Sen. *Predicting Tradeoffs in Contract-Based Distributed Scheduling*. PhD thesis, University of Michigan, December 1993.

[17] Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling: Preliminary results. In *Proceedings of the ACM Conference on Organizational Computing Systems*, pages 55–68, November 1991.

[18] Sandip Sen and Edmund H. Durfee. A formal analysis of communication and commitment in distributed meeting scheduling. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 333–342, February 1992.

[19] Sandip Sen and Edmund H. Durfee. Dependent subtask processing in a contract-net for manufacturing. In *Proc. of AAAI-93 Workshop on Intelligent Manufacturing Technology*, pages 7–13, July 1993.

[20] Sandip Sen and Edmund H. Durfee. On the design of an adaptive meeting scheduler. In *Proc. of the Tenth IEEE Conference on AI Applications*, pages 40–46, March 1994.

[21] Sandip Sen and Edmund H. Durfee. The role of commitment in cooperative negotiation. *International Journal of Intelligent and Cooperative Information Systems*, 3(1):67–81, 1994.

[22] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[23] K. Sugihara, T. Kikuno, and N. Yoshida. A meeting scheduler for office automation. *IEEE Transactions on Software Engineering*, 15(10):1141–1146, October 1989.

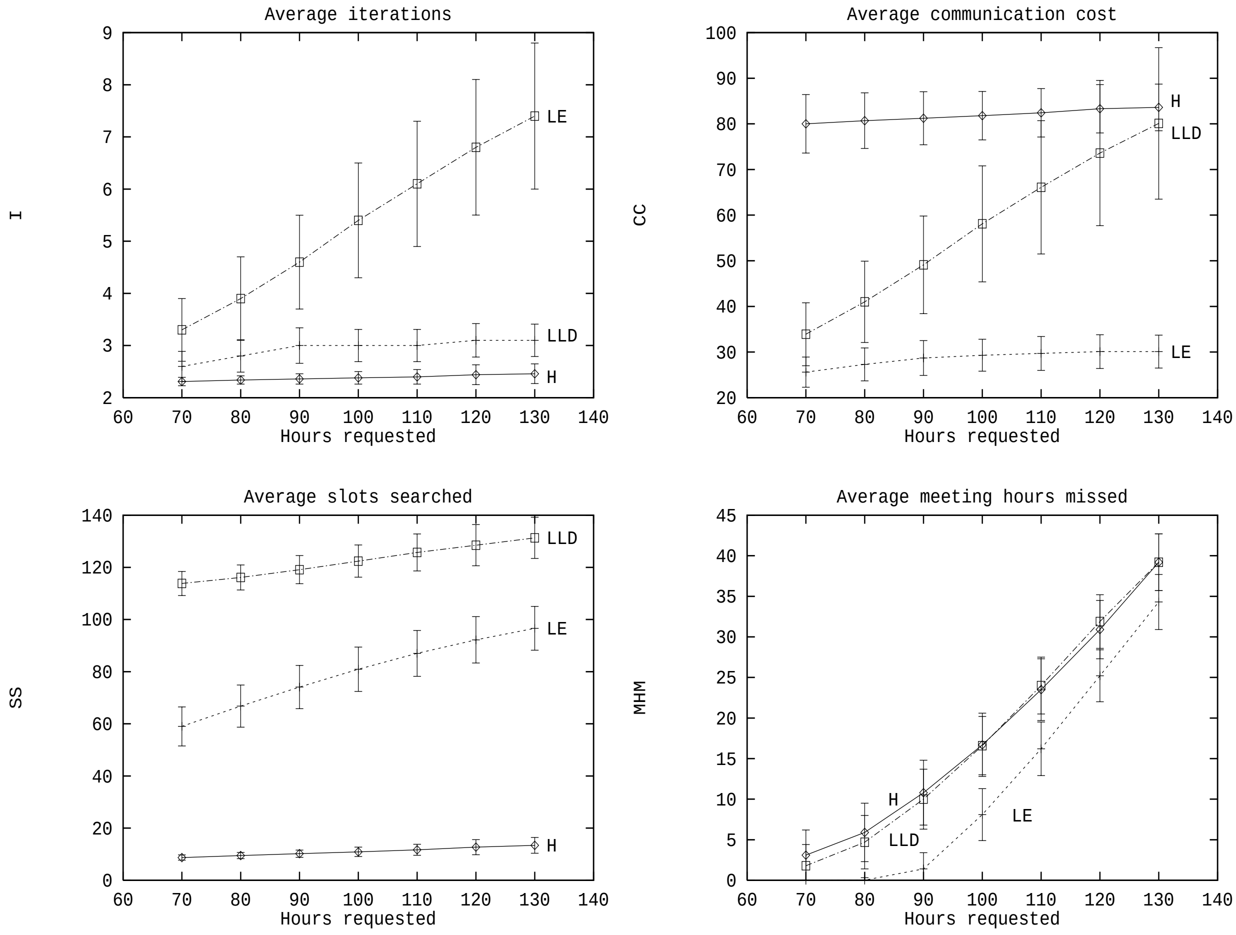

Figure 2: Average and standard deviation of the **LE**, **LLD**, and **H** search biases on the metrics evaluating the scheduling cost (**MHM**,**CC**, **I**, and **SS**) as $\mathcal{H}$ varies from 70 to 130.

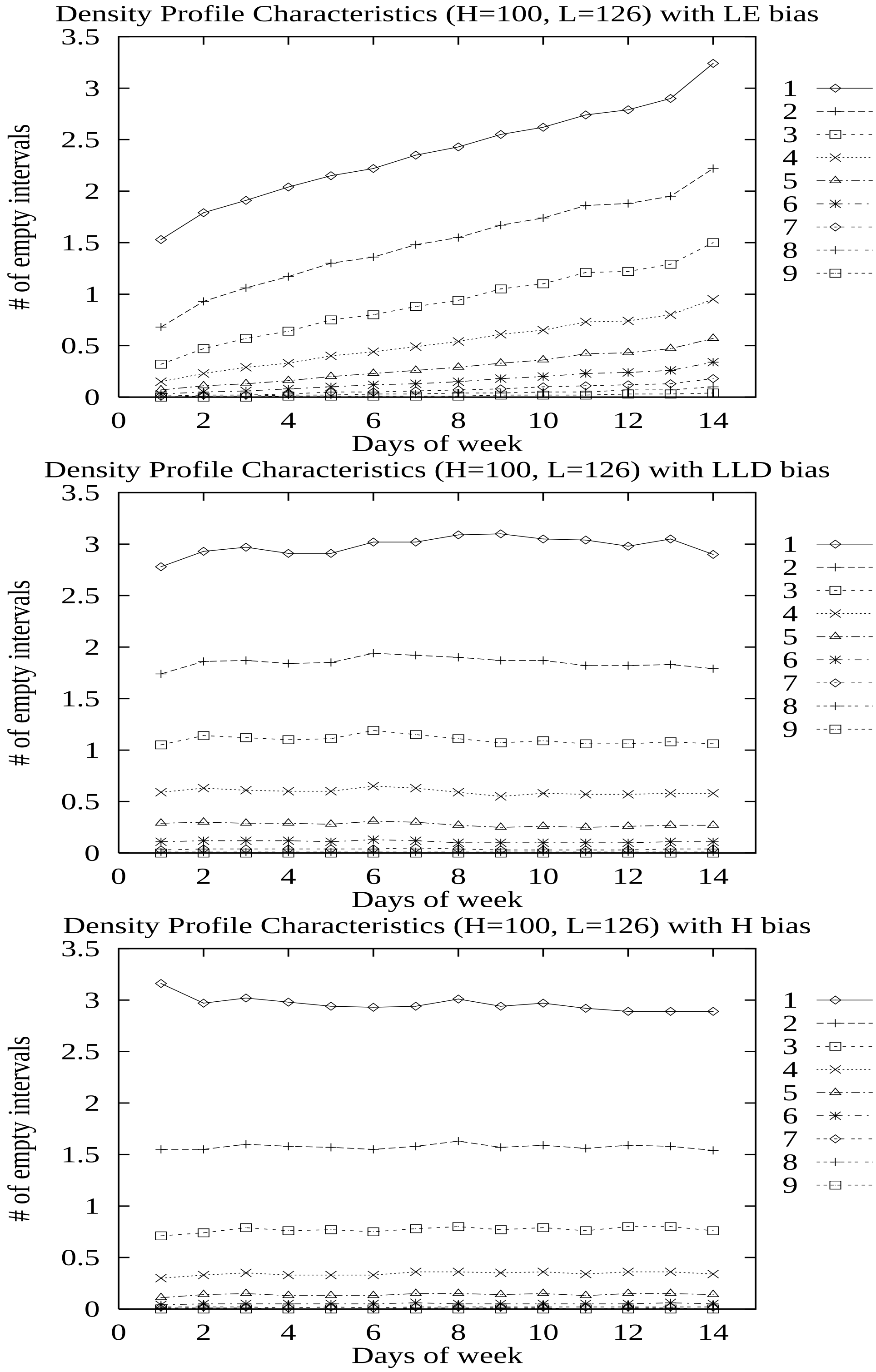

Figure 3: Performance of the **LE**, **LLD**, and **H** search biases (from left to right) on the metric evaluating quality of schedule generated (**DPC**) for the case where $\mathcal{H} = 100$.